Sudoku Solver

Naveen Kumar Dubey

VIT Chennai

3.1. Introduction

Backtracking algorithm tries to solve the puzzle by testing each cell for a valid solution.

If there's no violation of constraints, the algorithm moves to the next cell, fills in all potential solutions and repeats all checks.

If there's a violation, then it increments the cell value. Once, the value of the cell reaches 9, and there is still violation then the algorithm moves back to the previous cell and increases the value of that cell.

It tries all possible solutions.

3.2. Solution

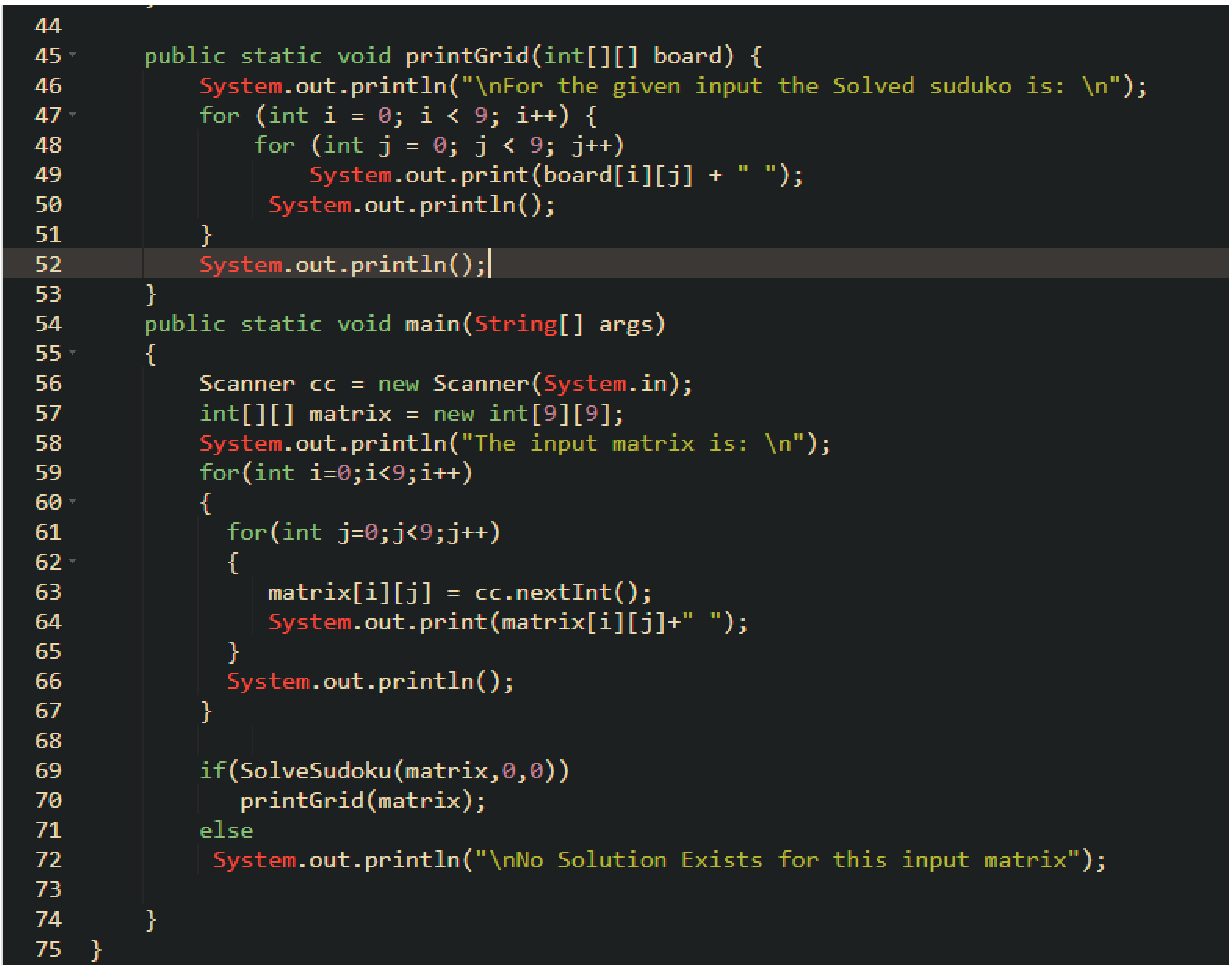
First of all, let's define our board as a two-dimensional array of integers. We will use 0 as our empty cell.

Approach: The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned position check if the matrix is safe or not. If safe print else recurs for other cases.

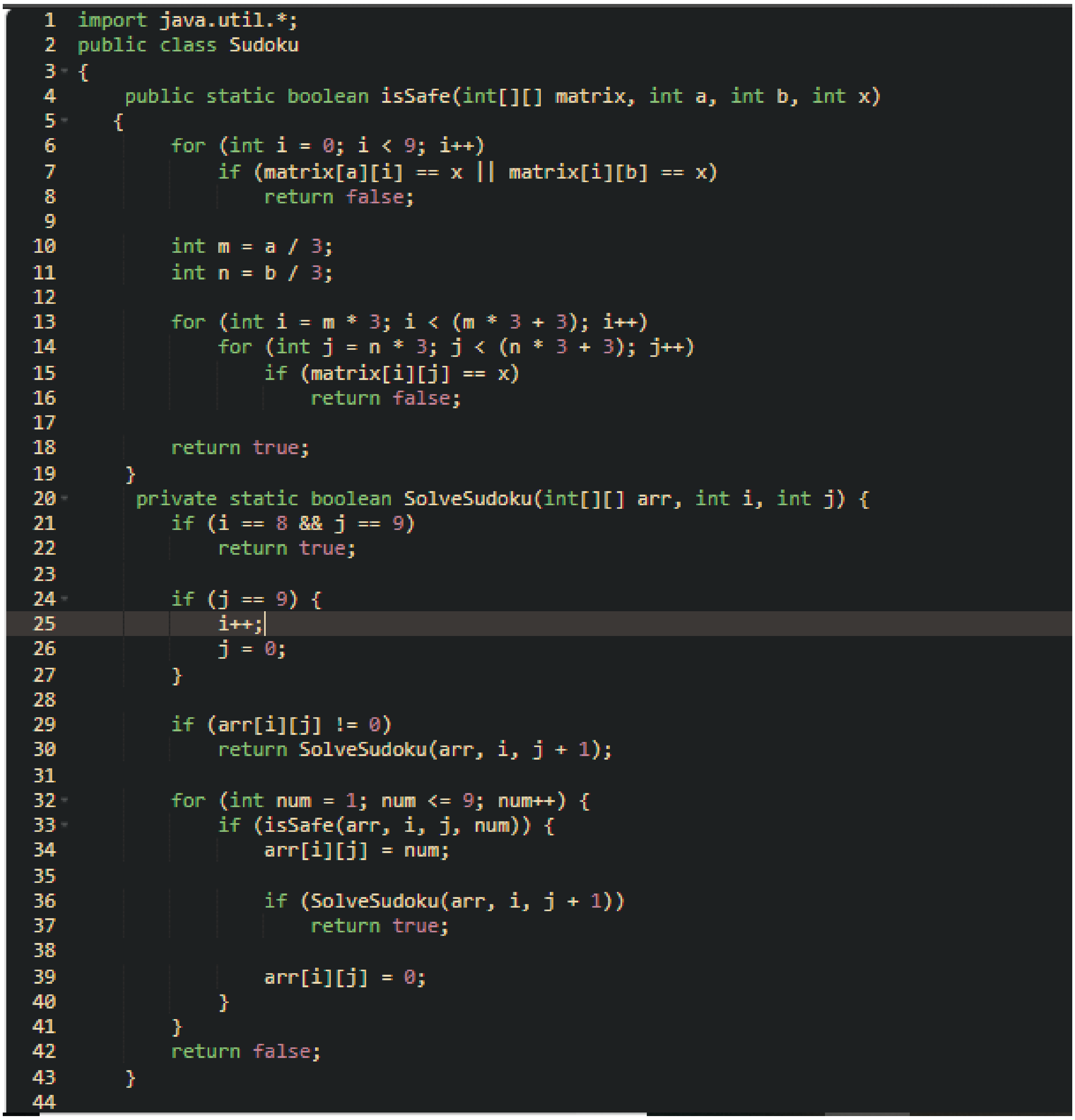
Algorithm:

1. Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true;
2. Create a recursive function that takes a grid and the current row and column index.
3. Check some base cases. If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false. The other base case is when the value of column is N, i.e j = N, then move to next row, i.e. i++ and j = 0.
4. if the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e. i, j+1. if the recursive call returns true then break the loop and return true.
5. if the current index is assigned then call the recursive function with index of next element, i.e. i, j+1

Entire code Link : https://www.jdoodle.com/embed/v0/2Ecp

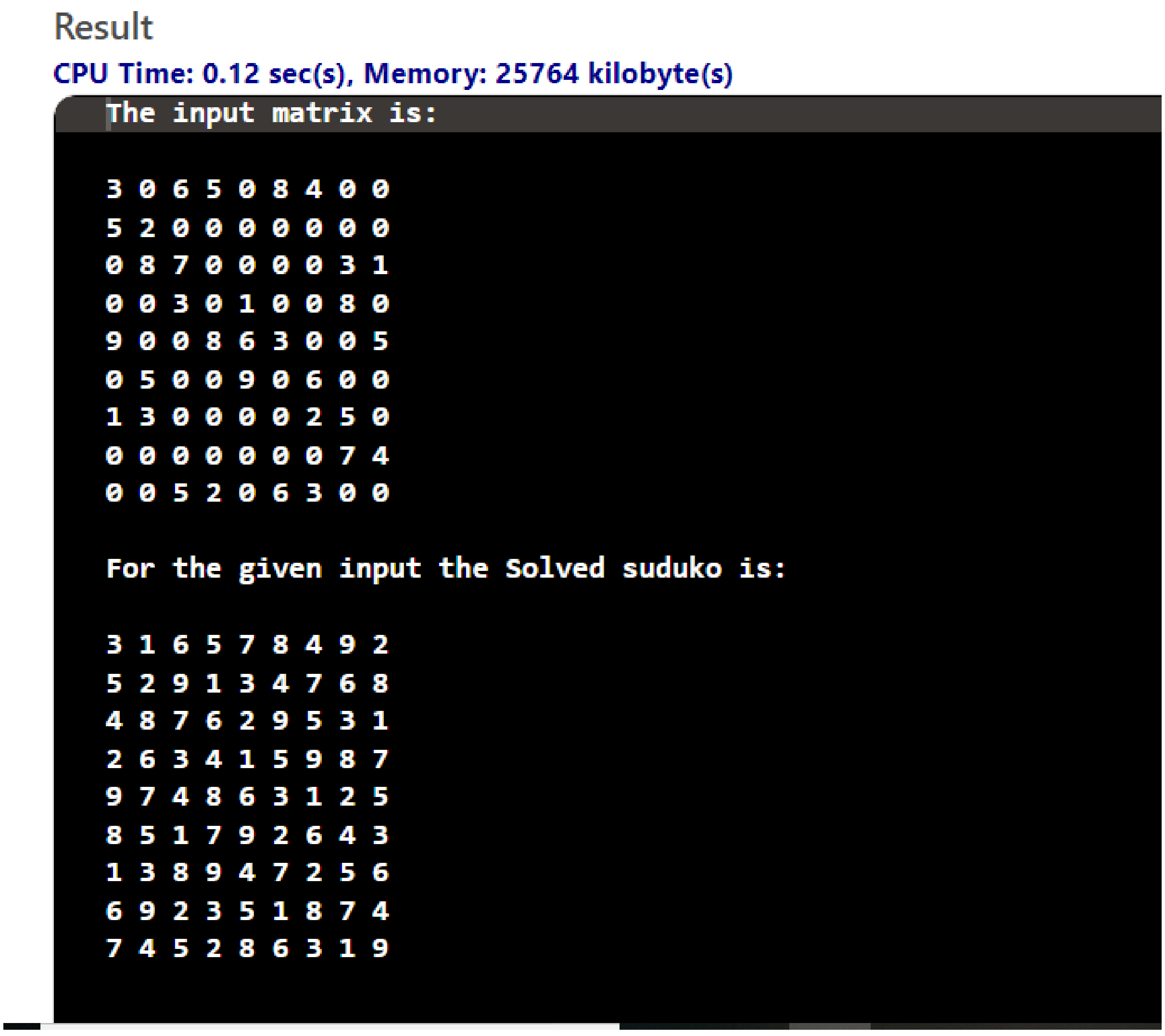


```

    public static void printGrid(int[][] board) {
        System.out.println("\nFor the given input the Solved suduko is: \n");
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++)
                System.out.print(board[i][j] + " ");
             System.out.println();
        }
        System.out.println();
    }
    public static void main(String[] args)
    {
        Scanner cc = new Scanner(System.in);
        int[][] matrix = new int[9][9];
        System.out.println("The input matrix is: \n");
        for(int i=0;i<9;i++)
        {
          for(int j=0;j<9;j++)
          {
             matrix[i][j] = cc.nextInt();
             System.out.print(matrix[i][j]+" ");
          }
          System.out.println();
        }

        if(SolveSudoku(matrix,0,0))
           printGrid(matrix);
        else
         System.out.println("\nNo Solution Exists for this input matrix");

    }
}
```



```java
import java.util.*;
public class Sudoku
{
     public static boolean isSafe(int[][] matrix, int a, int b, int x)
    {
          for (int i = 0; i < 9; i++)
              if (matrix[a][i] == x || matrix[i][b] == x)
                  return false;

          int m = a / 3;
          int n = b / 3;

          for (int i = m * 3; i < (m * 3 + 3); i++)
              for (int j = n * 3; j < (n * 3 + 3); j++)
                  if (matrix[i][j] == x)
                      return false;

          return true;
     }
      private static boolean SolveSudoku(int[][] arr, int i, int j) {
          if (i == 8 && j == 9)
              return true;

          if (j == 9) {
              i++;
              j = 0;
          }

          if (arr[i][j] != 0)
              return SolveSudoku(arr, i, j + 1);

          for (int num = 1; num <= 9; num++) {
              if (isSafe(arr, i, j, num)) {
                  arr[i][j] = num;

                  if (SolveSudoku(arr, i, j + 1))
                      return true;

                  arr[i][j] = 0;
              }
          }
          return false;
     }

```

Output :



Result

CPU Time: 0.12 sec(s), Memory: 25764 kilobyte(s)

```
The input matrix is:

3 0 6 5 0 8 4 0 0
5 2 0 0 0 0 0 0 0
0 8 7 0 0 0 0 3 1
0 0 3 0 1 0 0 8 0
9 0 0 8 6 3 0 0 5
0 5 0 0 9 0 6 0 0
1 3 0 0 0 0 2 5 0
0 0 0 0 0 0 0 7 4
0 0 5 2 0 6 3 0 0

For the given input the Solved suduko is:

3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
```

Thank You